Implementation of Perception Stack in F1 Tenth Cars

Akash Sundar

April, 2023

1 Stage 1: Getting started with F1 Tenth Cars

1.1 Getting started with ROS

Completed Lab 1 of the F1 Tenth course. Link
includes writing a publisher and subscriber model to subscribe to and relay vehicle parameters.

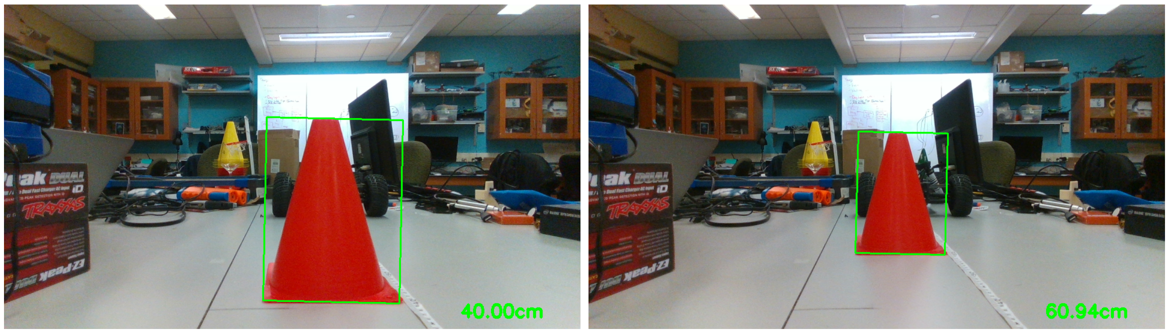
1.2 Getting started with Jetson Nano and RealSense 3D camera

Set up the hardware and learnt to use it to capture and process images.

1.3 Getting started with perception for F1 cars

Completed Lab 8 of the F1 Tenth course.

distance.py
Defined a function to identify and measure the distance to a cone in an image. Link



Algorithm:

1. Calibrate the camera
2. Convert image to hsv and identify colour threshold of cone
3. Perform erosion, dilation and blurring to remove noise
4. Find canny edge map of image and hence find contours
5. Simplify contours using polynomial approximations and convex hull functions
6. Identify primary/ largest contour by area
7. Find smallest rotated rectangle capable of circumscribing the contour using minAreaRect function
8. Find the vertices of the rectangle using boxPoints function
9. Use knowledge of environment to identify depth of cone

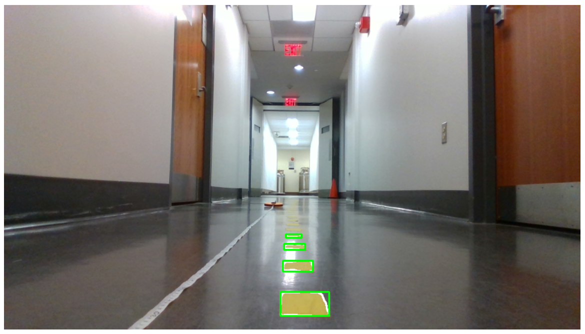
lane.py

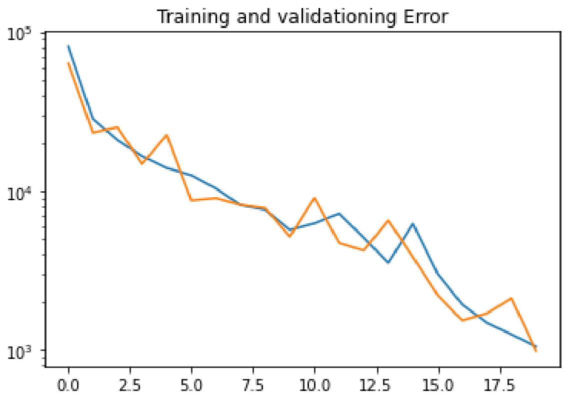
Defined a function to identify lane marking from an image.
Algorithm similar to distance.py. Link

yolo.ipynb

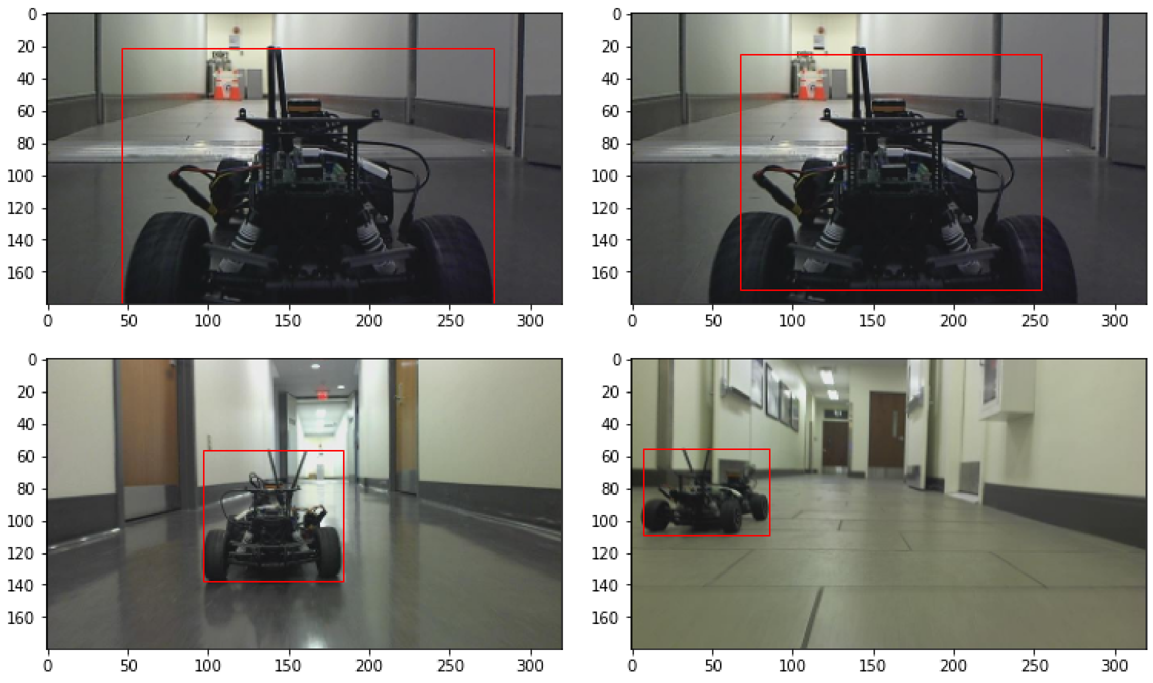
Defined a neural network to correctly identify and bound images of F1 tenth cars with over 91 % accuracy.
Link

The training loss vs validation loss has been plotted as follows. The learning rate, batch size and number of epochs were fine tuned to prevent overfitting and underfitting.





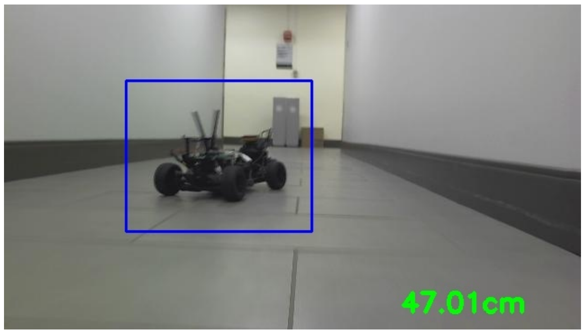
A few sample objects detected are displayed as follows:



1.4 Deployment of Models with Jetson Nano and Real Sense D435i and Evaluation

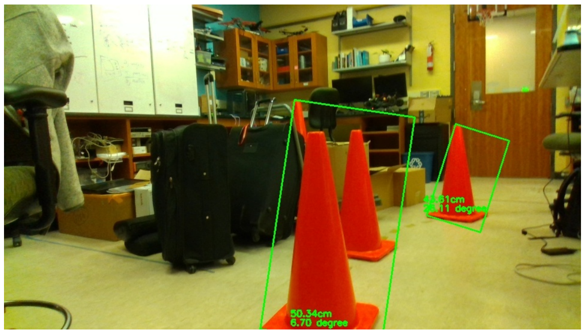
Deployed the onnx models built using a TensorRT engine to detect real instances of the objects. Model was required to be further fine tuned and minimized in order to account for the limited memory of the Jetson Nano.

However the actual final output proved to be less than acceptable. Possible reasons could include cleanliness of training data which was not reproduced in testing data. The car in the test cases were often present among highly complex backgrounds that could've caused the classification to disrupt. This facilitated the need for a more robust object detection algorithm as if the car was to be completely dependant on the inputs from its vision system, it was important to take into account the contingencies when people or other objects could be in or around the track and could lead to poor model performance.



Similarly, the classical cone detection model was integrated into the object detection pipeline and was also deployed on the Jetson Nano. Memory was not a major consideration anymore and this approach provided for a quicker detection with lesser frame lag. The algorithm worked perfectly fine in the detection of a single cone in the view. However, it had the demerits inherent in any classical vision approach. Namely, the very algorithm focused on identifying the cones based on color. Therefore, when multiple cones were present, with parts of one cone hidden behind the other, the algorithm naively creates a contour common to both the cones and draws a single bounding box around them. This inherently affects the data obtained from the environment.

In addition to the previous measurement of distance to the bounding box, the effective angle to the bounding box was also measured taking the normal direction to the plane to be 0 °.



2 Stage 2: Development of Robust Object Detection Algorithm

2.1 Classical Vision Based Approach

As observed in the previous section, cones were easily identified using classical vision methods. This approach aimed to identify and improve upon the inherent problems with the model.

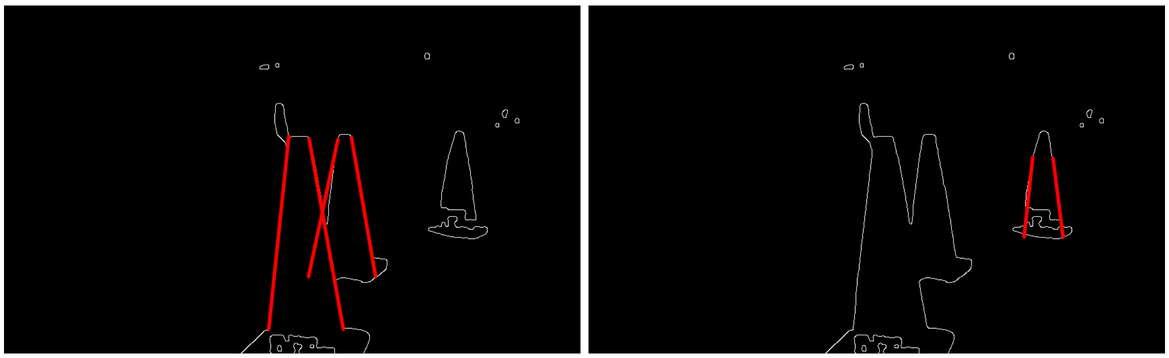
The main aim was to detect the two sides of the cone and represent each cone to be the area bounded by the 2 sloping sides. The tool identified for this approach was the Hough Transform to identify straight lines in the world.

The hough transform is heavily dependent of the canny edge detection algorithm. The edge detection algorithm as before, suffered from the problem of omitting edges when one cone obstructs the view of another cone of the same color. Also, a naive implementation of the hough transform returns multiple broken lines for each edge. Hence the output of the hough transform needed to be processed to make the lines fit more cloesly to the sides of the cone and make it a viable tool.

Algorithm:

1. Generate edge map using Canny Edge Detection algorithm
2. Apply Hough Transform on the map
3. Fine tune the parameters of the hough transform until best possible output is reached
4. Merge the multiple lines
5. Use the initial estimate of bounding boxes to remove line outliers that did not belong to any cone
6. Extend the lines to make the magnitude of both sides of the cone equal
7. Find best cone approximation

RANSAC was used in the merging of lines and finding the best approximation. The merged line was taken to be the combination of lines in produced by the hough transform such that a maximum number of points on the cone lay on the edge produced by the edge detection algorithm. The extension of the line simply elongated the line in both directions to match the length of its pair. However, this was not always the best line that fit the cone. Hence a threshold was provided to allow for rotation of the line to find the best possible approximation that had maximum inliers on the edge map.

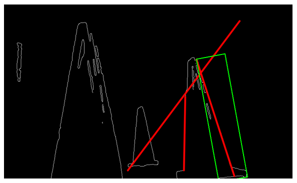


Drawbacks of current approach:

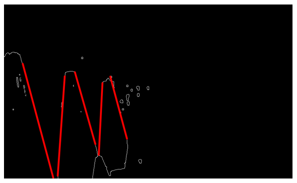
Although the approach worked perfectly well for the image above, it suffered from a loss of generalisation. As mentioned previously, the transform depends highly on the detected edges. A simple change in lighting could result in a shine on the cones which could adversely change the output.

Despite attempts at processing the output, it was often still difficult to obtain a 1 to 1 mapping for each cone and a pair of lines.

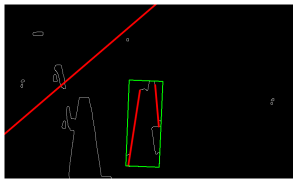
When the overlap between the cones is very high and the common edge is completely hidden (i.e., tip of one cone is often obstructed), no amount of processing is able to create an edge that does not exist. A few figures representative of these cases have been provided below.



Change of Illumination



1 to 1 Mapping of Edges



Tip Obstructed

2.2 Deep Learning Based Approach

The YOLO model used to detect instances of the car was one of the most basic models in vision. I explored alternate models such as MobileNet, Faster RCNN etc in an attempt to detect the object better.

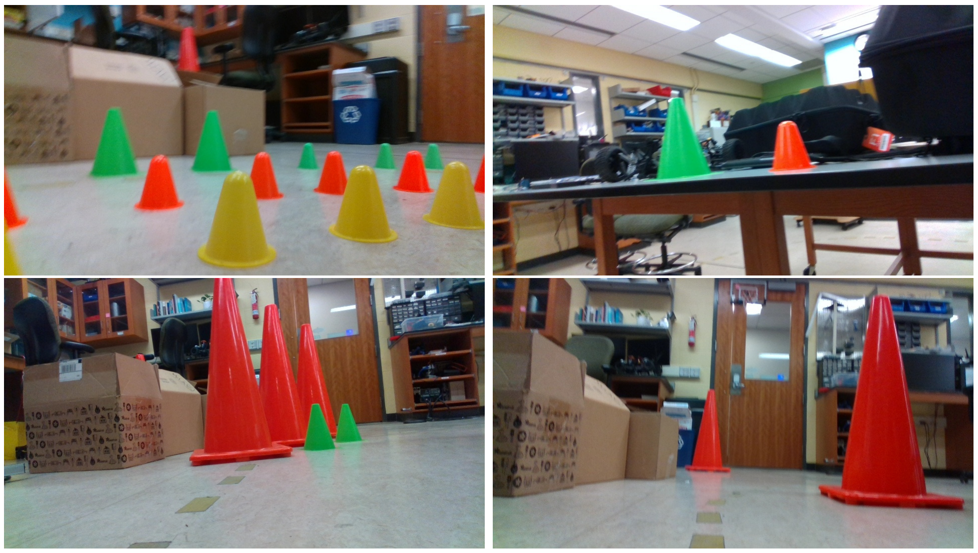
Similarly, a second YOLO model was also built to detect cones by training upon a dataset of 300 images lifted off the web and manually annotated.

However, both these models still suffered from generalisation. It was important to consider that the model was being deployed in a noisy environment and this noise needed to be reflected in the image. It was also observed that the images supplied to the model were not always of the highest quality. The speed of movement of the car could often produce lossy images that further add noise. Careful avoidance of any other objects in the field of view and using a suitable height of camera and angle of elevation often produced good results, but with the existence of such an informative description of the environment, no pipeline is even required in the first place. Hence a more representative dataset was required in order to better learn the objects.

2.3 Dataset Improvement

The dataset obtained off the web was often HD images with clearly apparent cones. I manually collected an additional 100+ images by using various arrangements of cones of varied sizes and colors and categorically eliminated certain instances of noise by considering different orientations of the same objects. Careful consideration was also taken to include some lossy images in the dataset so that the model could learn to account for the noise too.

A few examples of the images obtained are shown below.



The poor accuracy of detection can be attributed to the pursuit of a robust algorithm. This problem can be rectified by better defining the problem statement to include only one type of cone of a certain height viewed from a certain elevation angle. This places limitations on the use case but greatly narrows the corner cases to be explored when detecting cones.